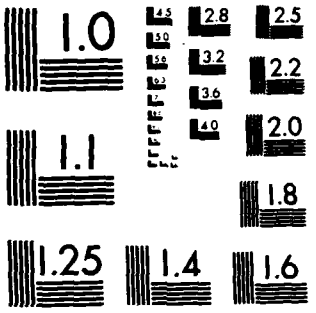
AD-A138 937 SOFTWARE ENGINEERING FOR USER INTERFACES(U) CALIFORNIA 1/1
UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE
S W DRAPER ET AL. JAN 84 ICS-8401 N00014-79-C-0323
UNCLASSIFIED F/G 9/2 NL

END
DATE FILMED
4 84
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12)

AD A138937

COGNITIVE SCIENCE

UCSD

INSTITUTE FOR COGNITIVE SCIENCE

UNIVERSITY OF CALIFORNIA, SAN DIEGO LA JOLLA, CALIFORNIA 92093

84 03 13 017

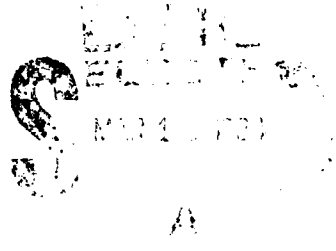
(12)

ICS REPORT 8401

SOFTWARE ENGINEERING FOR USER INTERFACES

Stephen W. Draper and Donald A. Norman

INSTITUTE FOR COGNITIVE SCIENCE
UNIVERSITY OF CALIFORNIA, SAN DIEGO
LA JOLLA, CALIFORNIA 92093



A

The research reported here was conducted under Contract N00014-79-C-0323, NR 667-437 with the Personnel and Training Research Programs of the Office of Naval Research, and was sponsored by the Office of Naval Research and a grant from the System Development Foundation. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsoring agencies. Approved for public release; distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the United States Government.

ONR REPORT 8401

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
~~ONR~~ 8401	2. GOVT ACCESSION NO. AD-A138 937	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Software Engineering for User Interfaces		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Stephen W. Draper and Donald A. Norman		8. CONTRACT OR GRANT NUMBER(s) N00014-79-C-0323
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Human Information Processing Institute for Cognitive Science University of California, San Diego La Jolla, CA 92093		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 667-437
11. CONTROLLING OFFICE NAME AND ADDRESS Personnel & Training (Code 442) Office of Naval Research 800 No. Quincy St., Arlington, VA 22217		12. REPORT DATE January, 1984
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

This research was also supported by a grant from the System Development Foundation.

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

software engineering	benchmarks	user needs
human-machine interface	debugging	acceptance tests
design	tradeoffs	
interface evaluation	documentation	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

OVER

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102 LF 014 6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT

The discipline of Software Engineering can be extended in a natural way to deal with the issues raised by a systematic approach to the design of human-machine interfaces. Two main points are made: that the user should be treated as part of the system being designed, and that projects should be organized to take account of the current (small) state of a priori knowledge about how to design interfaces.

Because the principles of good user-interface design are not yet well specified (and not yet known), interfaces should be developed through an iterative process. This means that it is essential to develop tools for evaluation and debugging of the interface, much the same way as tools have been developed for the evaluation and debugging of program code. We need to develop methods of detecting bugs in the interface and of diagnosing their cause. The tools for testing interfaces should include measures of interface performance, acceptance tests, and benchmarks. Developing useful measures is a non-trivial task, but a start can and should be made.

SOFTWARE ENGINEERING FOR USER INTERFACES*

Stephen W. Draper
and
Donald A. Norman

Institute for Cognitive Science C-015
University of California, San Diego
La Jolla, California 92093

ABSTRACT

The discipline of Software Engineering can be extended in a natural way to deal with the issues raised by a systematic approach to the design of human-machine interfaces. Two main points are made: that the user should be treated as part of the system being designed, and that projects should be organized to take account of the current (small) state of a priori knowledge about how to design interfaces.

Because the principles of good user-interface design are not yet well specified (and not yet known), interfaces should be developed through an iterative process. This means that it is essential to develop tools for evaluation and debugging of the interface, much the same way as tools have been developed for the evaluation and debugging of program code. We need to develop methods of detecting bugs in the interface and of diagnosing their cause. The tools for testing interfaces should include measures of interface performance, acceptance tests, and benchmarks. Developing useful measures is a non-trivial task, but a start can and should be made.

Introduction

The subject of this paper is the extension of Software Engineering to deal with the issues raised by the design of human-machine interfaces. To a large extent, all that is needed is to take the problem of engineering the user interface as seriously as any other part of software engineering and to apply to it the same kind of techniques, appropriately adapted. For instance, although the interface is implemented *in software*, it can be thought of as being "run" on human users. This means that we must modify our concept of a program *bug* to allow for part of the system to be a person; we must establish new performance criteria for the combined human-plus-interface system. Much of the thrust of this paper is simply to draw analogies with existing software practices in order to suggest how to support a professional approach to interface design. We do not present detailed ideas on what interfaces should be like,** but rather sketch some consequences for software engineering when interface design is taken seriously.

There are two themes in this paper: arguing by analogy with existing practices to their extension to interface design; and arguing from the nature of the problems of interface design to requirements for an appropriate engineering discipline. The first part of the paper makes some general points, the second summarizes their consequences for the coding, documentation, debugging, and testing phases.

Software Engineering for Human-Computer Interface Design

Two Goals in Optimizing an Interface

We begin with a tradeoff that has clear parallels to aspects of software engineering, the tradeoff between two quite different broad aims for an interface:

- Achieving speed and convenience of use (*power*) for the practiced user;
- Achieving ease of learning and use (*ELU*).

These distinctions are related to the familiar novice-expert distinction. There is a clear analogy between these aims and the desire to optimize space and time in software performance. In general it is desirable to optimize both aims, but beyond a certain point, the engineer must choose a particular tradeoff between the two. This analogy holds quite closely. Not only is there a reasonable region where one aim trades off against the other, but, at the extremes, the programs can be generally unusable.

* This research was conducted under Contract N00014-79-C-0323, NR 667-437 with the Personnel and Training Research Programs of the Office of Naval Research and by a grant from the System Development Foundation. Requests for reprints should be sent to the Institute for Cognitive Science C-015; University of California, San Diego; La Jolla, California, 92093, USA. We thank Tony Wasserman, Eileen Conway, and Sondra Buffett for their assistance with the manuscript.

** We leave out of this paper any major discussion of what a human-computer interface ought to be, or what the general problems or principles are, because this topic is explored in numerous other papers by us and by the growing community of people who work in the field called *Human-Computer Interaction*. This field now has an annual meeting, several journals, and an ACM Special Interest Group (SIGCHI): we see no need to cover the material here. See, for example, the *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems*.[1] This paper concentrates on the lessons that can be applied to interface design from practices already common in Software Engineering.

Consider the extremes. Suppose we optimized program speed or interface power. Just as a fast program that requires more space than any customer can afford is in practice useless, so too is an interface that provides immensely fast and "powerful" interaction, but at the cost of requiring such a level of skill that no user will actually be able to use it because of the difficulty of learning it presents. Similarly, if we optimized program space or ELU, the program might be too slow to use regularly (no matter how economical of space) and the interface can be too laborious for a regular user to employ productively, no matter how safe, "friendly," helpful, and easy to learn. If we avoid the extremes of these dimensions, then there is a wide range of choice for the designer. The existence of programs of similar function on microprocessors and large mainframe computers is a strong reminder that we can tradeoff both in the space-time domain for the software and in the power/ELU domain for the user interface.

User and Program as Communicating Co-Routines

When a programmer implements a system with a user interface, he or she is not only defining what the machine will do but also defining what the user can do. The program and user can be thought of as co-routines, each communicating with one another. The programmer must explicitly or implicitly decide what actions and information will be available to the user. For any given interface, this point can be illustrated by drawing a flow chart of the user's part of the process. Although this point seems simple, it changes the fundamental (though usually unacknowledged) situation by which programming takes place. No longer are programmers engaged in the private task of getting the machine to do something for themselves—they are not even engaged in communicating a program to other programmers, a view intermittently voiced by purist programmers. Instead, the interface designer should be viewed as someone who must write a successful co-routine between user and machine, yet where the full specification of the user side of the co-routine is not well known (and may be variable): this is the real subject of the design, and this must become the conscious objective.

There are two consequences to this point, both discussed in greater detail later in the paper. First, we need languages that support this view, that is, that represent this co-routine directly. Second, the interaction needs testing and debugging on typical "hardware," hardware that includes representative users.

The State of User Interface Design

How much is known about how to design interfaces? We must answer this question in order to plan a sensible software engineering strategy. If enough is known to lay down detailed principles, then a top-down strategy might be followed; otherwise an iterative strategy must be adopted with emphasis on the testing and debugging phases.

Quantitative principles. Top-down principles of design should, in the ideal case, allow a design to be worked out in advance rather than entirely by trial and error. These tools need to be developed from a solid basis in experimental psychology, coupled with a good understanding of programming and software design. Not surprisingly, the number of groups capable of this work is limited and not much yet exists. There are now several initiatives whose goal is to provide quantitative principles for the design of human-computer interfaces. For example, one of us has shown how it is possible to develop a quantitative assessment of how design tradeoffs affect the *User Satisfaction* for an interface[2]. Thus, in trading of time, information, and workspace, it is possible to compute the tradeoff space, showing the psychological impact of tradeoffs in these variables. A complementary approach is that of Card, Moran, and Newell[3] who provide a set of quantitative tools for computing the operational parameters of interfaces.

The development of psychologically based, quantitative design tools is still in its infancy and much work remains to be done, both in development and in validation. If we can develop sufficient range and breadth of quantitative design tools, then we can look forward to a time when it will be possible to provide general principles and even tables of numerical values in design handbooks, allowing such design considerations as workspace size, display time, menu structure, command language design, pointing-versus-naming, interface power, and ELU to be assessed at the design stage, without the need to build a test system. For the present, however, the designer, by and large, does not have quantitative data at hand.

Qualitative principles. In the absence of well-established quantitative design aids, we need methods that allow us to work with qualitative principles. A large number of qualitative principles exist, many in the form of "slogans," exhorting the designer to consider this factor or that, or to avoid this failing or that [4,5,6,7]. These qualitative rules are often quite reasonable (which makes it especially discouraging to see how frequently even the most obvious and elementary principles are violated in existing systems), although if the design of a user interface is ever to proceed in a systematic fashion, we need to go beyond this level.

One major example of the attempt to base a system design on fundamental design principles, built in such a way as to capitalize on the user's existing knowledge is the design of the Xerox *Star* system[8]. This design attempted a systematic strategy based on principles of good human-computer interaction. Since then, of course, other systems have been developed along similar principles (in particular, the Apple *Lisa* system). The *Star* illustrates the delicate tradeoff decisions that must be faced: the attempt to optimize ELU led to degradation of performance, especially in speed of the system, as well as to a high selling-price.

Specifications for Interfaces

In defining and addressing a software project two questions must be tackled: what kind of specifications are reasonable for defining the project?, what kind of divisions of the project into subtasks are likely to be viable? The same observation applies to both: specifications are determined at an early stage and must remain constant if massive re-writing is to be avoided.

Sheil[9] argues against the utility of a Structured Programming approach to user interface design. Sheil points out that whatever the virtues of that approach, it depends on having a clear and fixed specification at the start. The technique becomes strained whenever the specifications change during the life of the software. Although in the ideal case a Structured Program practitioner would re-do the whole design and implementation at every change in specification, in practice the investment in the existing code and design exerts an enormous pressure toward piecemeal change, i.e., toward iterative redesign. In areas where such *shifts in specification are common*, it seems better to face the fact that development will in practice be iterative. Sheil suggests—and we agree—that interface design is one such area.

Several consequences follow from this point of view. *First, it will be unwise to* allow any details of a user interface to appear in the project specifications: for instance instead of giving a list of the commands to be implemented, specifications should rather be of the form discussed by Shneiderman[10], who suggests instituting acceptance tests for interfaces such as "after 75 min. of training 40 typical users should be able to accomplish 80% of the benchmark tasks in 35 min. with fewer than 12 errors." A professional approach to user interface design puts the responsibility with the designer, having the user or customer specify performance requirements rather than details, just as in Shneiderman's hypothetical acceptance test. This leaves the design of the interface to the software team, but with explicit standards for the specification of the performance of the system.

A second consequence is that we must expect to have to go through many iterations on details of the interface. If the overall project is to be subdivided by dividing the system into modules that are separately designed and implemented, then a third consequence is that the user interface should be segregated into a single module which all other modules must use if they communicate with the end user.

Separating interface design from program design. This strategy consists of isolating the main program in one set of modules, the user interface in another, and considering the user as comprising a third:

The virtue of this view is well known in the software engineering community: as long as the communication structure and the data representation are well specified and adhered to, the modules can be worked on independently and changed at will, without any effect on the rest of the system. The *communication between the Interface* and the *User* can be changed in any arbitrary manner as long as the communication between the *Program* and the *Interface* is not affected. This requires, of course, that the only part of the system that may interact directly with the user is the *Interface Module*. A well-defined protocol between the main program and the interface module can be defined at the outset, leaving the interface designer free to change the interface without interfering with the independent development of the main program. An important benefit of this modularization strategy is that it allows separate specialists to work on the main program and on the interface module.

A good example of this modularization combined with support for a heavily iterative approach to developing the user interface is provided by the database and interface packages *Troll/USE* (the database)[11] and *RAPID/USE* (the interface package)[12] *Troll/USE* is a relational database—the "main program" module. *RAPID/USE* is a tool for the design of interactive dialogues and systems, allowing for rapid modification and experimentation of the interface (through a transition network specification), yet maintaining constant the necessary communication protocol to the database, *Troll/USE*. Clearly the design of a database requires different skills than designing a user dialogue, and in fact *RAPID/USE* is designed to allow non-programmers to design the dialogue using a simple interpreted language. Other examples are the DMS system[13] and the work by Buxton *et al.*[14]

Makeup of interface design teams. A further consequence of the special nature of interface design affects the organization of the design team. If the principles of interface design were developed to the point where their application required no personal expertise, and if the tradeoffs were all identified and quantified in advance, then a split between groups of experts might be workable. It is noticeable that some of the more successful interface designs (e.g., the Xerox *Star* and the Apple *Lisa*) have been done by teams which included people with diverse talents and training, but which did not make a distinction between evaluators and implementors, between psychologists and programmers. We interpret this to indicate that at the present time the closest cooperation is required to identify and resolve the unexpected tradeoffs that surface in the course of a design. Many design questions are at heart unforseen tradeoff decisions, and these can only be reasonably made by people who appreciate all the factors involved. This, although a separation of the interface from other program modules is highly beneficial, and although logically the skills required for designing each are quite different (e.g., dialogue design versus application programming), it seems that the unknowns in interface design continue to demand close cooperation between parts of design teams.

Influences of the Interface on Code

Languages for User Interface Design

We have argued that interface designers need to be aware that they are designing a dialogue, and more than that a co-routine, between user and computer. The idea immediately follows that special languages that represent this view would be an important aid. Certainly conventional languages are poor in this respect because they are machine-centered: they describe events around the sequence of machine actions with input and output seen as side-effects. Wasserman's *RAPID/USE* system offers the designer a language based on Transition Diagrams, where nodes correspond to a machine state and the output displayed at that point, and the various arcs from a node correspond to alternative user inputs. This is a major step in the right direction. It is perhaps not a complete solution because it is essentially a representation of the machine the user sees (as a quasi-finite state machine) rather than of the co-routine. The user's processes are only implicit (though much easier to see than in ordinary languages), and machine actions are represented only as side-effects of transitions. Until further advances in this area are made, exercises such as flow-charting the user's decision-making and actions will remain important.

The Interface to the User Module

We have argued that the user should be viewed as a module of the system. Good programming practices now demand that one establish a uniform communication protocol early in the design phase and stick to it. Part of the rationale is economy—if module U has to use one protocol for interacting with module A and another for module B, it will take more code to do so. (Or, by analogy, if you think of module U as being the User, then the user has more to learn.) In addition, there is scope for confusion and error in the programming if there is not a uniform protocol (in the analogy, the lack of consistency makes it harder for the user to learn and to apply the protocols).

These arguments apply best to low-level user protocols. The more programs use a given protocol for interaction, the more benefit the user gains from the associated set of skills in using it (i.e., from their associated "subroutines"). Thus a user is helped if all parts of the system use a small number of common protocols for interaction, protocols that the designer can think of as corresponding to subroutines in the user's mind.†

It is not sufficient to think of the user as a simple system module with a single protocol for interacting with other modules. There are many layers of protocol, just as there are in computer networking. At least three levels may be distinguished: (1) the low-level protocols just discussed; (2) the conceptual model of the domain the designer wishes to present to the user; and (3) the highest level where the user has several concurrent goals (e.g., to send a letter or get a budget analysis) and the computer is one means to the ends. Most of our knowledge about human-computer interaction is at the first two levels: little is understood/known about the highest level of user goals. The Xerox *Star*, the Apple *Lisa* and *Visicalc* can be seen as paradigm examples of the second level in that the system image was decided early on as a major design decision which defined the context in which the rest of the interface was developed.

Standardized Packages

A major development tool would be the creation of various packages for doing standard interface operations. There are two separate motivations for this:

- To provide a language at the right level for the designer (where the operations are elements of user interaction and lower-level details can be hidden);
- To provide standard modes of interaction across the system to help the user.

The first provision helps the designer to work on the design of the interface undistracted by implementation details. The second provision gives a standardization (consistency) across applications that helps the user, as argued above. Simply providing packages is often enough to get standardization—it makes it easier for a programmer to conform than to dissent. There is great need for software tools for interfaces, including screen management, user-program dialogue, and packages for doing help, argument parsing, history, and undo. A good example is Perlman's[15] general interface package.

Documentation as an Integral Part of the Interface

An obvious consequence of integrating user interface design into the overall software engineering is to integrate documentation and code generation. Mashey and his colleagues at Bell Labs have taken a major step in this direction by using the same file control system for both source code and documentation[16], thus promoting their simultaneous development and allowing immediate checks on whether one is out of date relative to the other. Knuth's recent work is along the same lines[17]. This is most likely to benefit other programmers (rather than users) who will have to maintain the code, since they are often the major beneficiaries of complete and correct documentation. However, if it is true that no interface design is likely to remain fixed for long, then it is not enough for it to be friendly to the user—it must be friendly to its maintainers as well.

The term "documentation" should not be viewed too narrowly. Users get information from a number of sources including manuals, tutorials, error messages, and normal

† A major problem here is to ensure a match between the actions expected of the user and the capabilities of that user. This is one of the major themes of research on human-computer interaction, and is a non-trivial problem. In general, we would submit that here is where the software designer must interact with a human-factors or psychology software designer — during the design phase — to develop the specifications of the user actions.

displays. One test of the adequacy of the overall documentation is to introduce an error in the operation of the system while observing a "typical" user in a "typical" situation. The observation of interest is to determine how the user copes with the situation: the design *fails the test if the user* cannot recover gracefully. It is important to note how the user determines the state of the system and the options that are available, and also to observe what the user actually does (which may be quite different from what the designer had in mind).

The success of the system will usually depend on a combination of information sources and is neither a property of the code alone nor of the documentation alone. The theme is that in order to provide good documentation from the user's point of view it is necessary to identify what information the user needs, and when. Then, it is necessary to provide a channel from the situation to the information. It is relatively unimportant what media are used for this in any given case: they could be messages generated by the system or notices stuck to the terminal. What matters is whether the user is able to get the information that is required. Note that it is not relevant that the information is available in principle. What matters is whether real users, in real situations, can get the answers. If the user cannot solve the problem, then it is the system design that is at fault, whether or not the designer can demonstrate that the relevant information was available to the user: the critical test is the practical one—do real users succeed at the task?

Debugging the Interface

When a piece of software has been implemented it needs to be tested and debugged: the same applies to the user interface. In the past, debugging the interface has generally been left to the customer, whose complaints are classified as changes to the specifications. The net effect tends to be that changes are slow, expensive, resented by the programming team, and do not benefit from any kind of systematic or professional approach. Clearly the field is more than ready for improved practices.

What Is a Bug?

The field of debugging involves many issues. One problem is to determine what counts as a bug, another is to determine what symptoms can be detected in practice (and what proportion of bugs escape because they produce no clear symptoms), and yet another to determine the cause from the symptom. The concept of "bug" is clearly useful in both traditional and interface software engineering, but nevertheless it has no clear definition. Some bugs are clear—*if* an explicit specification is not met, the implementation has a bug. However there can be bugs in the specifications themselves, and bugs relative to implicit specifications. A crucial part of developing interface engineering will be developing standards that become implicit specifications for all interface programming. (The analogous points for bugs in programs are discussed in Johnson, Draper, & Soloway[18].)

We believe that the system specifications should include a statement about the class of user and the kind of training that is to be expected. The system should then be evaluated with that very same class of user, with the same training procedure. If the user then has problems, there is a bug in the system. The bug could be in the training, or in the interface. The point is that we cannot determine just where the problem lies until we have explicit specifications for all aspects of the computer system, including the interface and the user performance. When we have specifications that cover the user, then we can determine how reasonable they are on the basis of the user's abilities. Only when we have determined that the specifications are indeed reasonable can we then claim that the system that fails to meet those specifications is at fault. This lesson applies to all parts of the system, of course, but its implications for assessing the role of the user as a part of the overall system operation seem not to be properly recognized.

Finding Bugs

The only way to find bugs is to test. This means that the system must be put through its paces with the human user, much as programs are put through their paces with test sets of data. Just as a program needs testing by data that exercise every branch of the code, so the user-program interaction needs testing by exercising each possible "branch" of the interaction. Unfortunately, test procedures for user interfaces do not exist.

Note that the testing phase is not apt to be easy. It requires the development of good test problems, of a good pool of users upon whom the tests will be run, and careful observation and evaluation of the result. It is critical that the users upon whom the system be tested reflect the actual user population for whom the system is designed. Psychology has amassed a number of methodological tools that can be of use. Other tools, specialized for this particular problem, need to be developed.

Learning how to ask users for information is as big a topic as learning how to extract useful measurements from computers. For instance, consider a faulty error message. If it is so useless that the user cannot understand it and gets stuck, there is often a bias against reporting the consequent failure to carry out the task successfully because the users are apt to feel that the problems are due to their own inadequacies. On the other hand, if the message is wrong or silly in some way but nevertheless the users succeed in diagnosing the real problem fairly quickly, then they are likely to express their irritation. Note that this means that the nonfatal inadequacy is likely to receive a much higher rate of spontaneous complaints than the much more serious case which causes users to fail completely. Obviously we need to learn how to work around phenomena like this. For instance, using exhaustive checklists in questionnaires[19] ensures that one solicits opinions on all parts of an interface, and, to some extent, allows one to see things such as mass avoidance of a command that no-one complains about spontaneously.

People are very sensitive to the context in which they are operating, and if one is not careful, the test population may feel that it is they who are being evaluated (rather than the system), and they may carefully monitor their responses and behavior so that they will not "look bad" or "stupid"[20]. One of us experienced the situation where a deficiency in the system was not reported by any of the users because they attributed the difficulty to their own inadequacies, not realizing that it could be avoided by a (rather simple) design change. In this case, it required an experienced observer to watch users and note the problem. Note also that the existence of any problem was at first denied by the design team who asked "but why has nobody ever complained?" This sounds reasonable, but is analogous to a programmer who does no systematic tests and then uses the length of time before the first complaint as evidence that complaints must be ill-founded. This is not a trivial instance: users who feel that a system reveals their inadequacies will not wish to use the system and will resist its introduction into the workplace. Thus, the system will not get used (or morale may suffer). The problem is to devise techniques that allow the designer to realize the nature of the difficulty. It will take extreme sensitivity on the part of the tester to overcome these problems. It is here that the skills of the experimental psychologist are probably essential.

Debugging Tools

The use of questionnaires is analogous to a post-mortem in that they are applied after the program has run. One of the most pressing needs for interface debugging is to have facilities analogous to run-time tests built into all computing environments that cause program exceptions for bad addresses, floating point overflow, etc. Although an important function for these is, of course, the protection of other users, they are also valuable for debugging because they stop execution at the earliest sign of trouble and give the programmer a chance to gather information on the state of the process at that point. Ease of debugging is crucially affected by the immediacy of error detection, as anyone knows who has debugged programs with and without array bounds checking. Applied to interface debugging, this means developing suitable error criteria, and then acting on it. It is not necessarily appropriate to halt a program when an error in interface interaction is detected, but at the least, one could create a relevant "dump"—a trace of the whole interaction together with as much information as possible on the users, their experience, and their current goals and thoughts at the time of the difficulty.

The various existing techniques for getting at the interaction between the user and the system differ in their immediacy and the information provided. Furthest removed from the actual interaction is the collection of opinions after some amount of experience with the system (e.g., at completion of the training period). Closer to the actual usage is the use of on-line complaint facilities. A still more immediate record is provided by history traces or dribble files that provide a detailed record of the low-level actions of the user, but without any of the goals or intentions. Intentions and goals can be gotten by the collection of real-time, thinking-aloud protocols from users while they interact with the system. Each technique offers a different perspective on the interaction.

Testing the Interface

Improving Measures of Performance

In addition to debugging, a programmer will typically be concerned with examining and optimizing certain measures. The best parallel here is with the problem of improving a program's speed of execution. The conventional wisdom on the timing problem is that a typical program spends 90% of its time in 10% of its code, so the strategy is to identify that 10% and work on tuning it, since work on improving the other 90% will show little effect overall. Thus, profiling tools that show where a program spends its time are important. In improving an interface, several issues are relevant: how many users find a given command problematic?, how problematic do they find it?, how often does the issue arise? As with debugging, we see here a gap between what can be easily and directly measured and the underlying concern of the designer.

Like profiling tools, then, interface tools should produce measures of those things that can be used by the designer to pick the next point of attack, together with a measure of how important it is to do any further improvements. Also like profiling tools, there will be issues of how accurate these measurements are (resolution difficulties) and how representative of the real situation. Ultimately a lot will also depend on the experience and judgment of the designer using the tool. Thus not only do the tools need to be developed, but it will then take a further significant amount of time to accumulate experience in the use of these measures.

Another tool is on-line command usage measurements. It is relatively easy to collect a running record of command use for the various users of a system, thus providing reliable measures of how often commands are used, and by what percentage of users. The frequency of use is important in weighing the priority to be given to problems.††

Benchmarks and Acceptance Tests

Earlier, we discussed Schneiderman's[21] suggestion that the specification of the interface be given in terms of the acceptance tests to which it will be subjected. This idea can have far-reaching effect in focusing designers' attention on a definite goal for the interface. Whether or not success at a particular test turns out to be a good guide to the user's long-term satisfaction with the product, it is at least at an appropriate level of specification; this is a crucial step in

†† There is a major problem of invasion of privacy. It is not appropriate to keep records on the details of individual users' interactions with a system. Our solution is to encode the user's identity so that although the user identity cannot be determined, we can still match up the particular command sequences and program usages with the user codes. This is essential in allowing the discovery of common patterns of operation.

amending software engineering to interface design.

User-interface benchmarks will be most clearly useful when the aspects of performance and the situation in which it is to be measured are clearly defined. As a general method by which to judge a whole system, benchmarks are obviously limited; systems differ on many dimensions and benchmarks often generate only a single measure. The use of benchmarks for interfaces is further problematic in these early days since we do not yet know all the crucial variables. For instance, discussions about which of two operating systems are more effective for a class of users are sometimes carried on without considering the communication rate of the channel to the user, yet this crucially affects how much feedback is perceived as a painfully time-wasting nuisance. In general, factors not directly under the control of the engineer may have a dominating effect. Until we are more confident of being conscious of the factors that have a major influence on the measures we are interested in, we will not know how to run benchmarks in which they are held constant.†††

[1] A. Janda, Ed., *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems*, Boston, Dec. 1983.

[2] D. A. Norman, "Design Principles for Human-Computer Interfaces," *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems*, pp. 1-10, Boston, Dec. 1983.

[3] S. Card, T. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Erlbaum, 1983.

[4] A. Badre and B. Shneiderman, Eds., *Directions in Human-Computer Interaction*. Norwood, NJ: Ablex, 1982.

[5] J. Nievergelt, "Errors in Dialog Design and How to Avoid Them," International Zurich Seminar on Digital Communications, IEEE, *Institut fuer Informatik, ETH*, 47, 1982a.

[6] J. Nievergelt, "Towards the Intergrated Interactive System: An Experiment in Man-Machine Communication," *Institut fuer Informatik, ETH*, 47, 1982b.

[7] B. Shneiderman, "Software Psychology: Human Factors in Computer and Information Systems," Cambridge, MA: Winthrop, 1980.

[8] D. C. Smith, C. Irby, R. Kimball, and B. Verplank, "Designing the *Star* User Interface," *Byte*, 7, pp. 242-282, April, 1982.

[9] B. Sheil, "Power Tools for Programers," *Datamation*, 29, pp. 131-144, Feb. 1983.

[10] B. Shneiderman, "The Future of Interactive Systems and the Emergence of Direct Manipulation," *Behavior and Information Technology*, 1, pp. 237-256, 1982.

††† A statistical problem arises in benchmark tests with users that does not normally arise with hardware: unlike computer hardware one can neither get identical people (so that single measurements generalize reliably) nor run a test twice on the same people with identical results (because of learning effects). Even when we understand the causes of variation well enough to apply statistics with confidence, this will still mean running large numbers of trials where one would have been sufficient to benchmark a machine.

[11] M. L. Kamlet, A. I. Wasserman, and R. P. van de Riet, "Troll/USE Reference Manual," Laboratory of Medical Information Science, University of California, San Francisco, Dec. 1982.

[12] A. I. Wasserman and D. T. Shewmake, "Rapid Prototyping of Interactive Information Systems," *Proceedings of the 2nd SIGOFT Symposium - Workshop on Rapid Prototyping*, Columbia, MD, 1982.

[13] J. Roach, H. R. Hartson, R. W. Ehrich, T. Yunte, and D. H. Johnson, "DMS: A Comprehensive System for Managing Human-Computer Dialogue," *Proceedings of the Human Factors in Computer System Conference*, pp. 102-105, Gaithersburg, MD, 1982.

[14] W. Buxton, M. R. Lamb, D. Sherman, and K. C. Smith, "Towards a Comprehensive User Interface Management System," *Computer Graphics*, pp. 35-41, 1983.

[15] G. Perlman, "Software Tools for User-Inteface Development," Presented at the Summer USENIX Conference, Toronto, Canada, 1983.

[16] M. Bianchi, R. Glushko, and J. Mashey, "A Software/Documentation Development Environment Built from the UNIX Toolkit," in H. J. Schneider and A. I. Wasserman, Eds., *Automated Tools for Information Systems Design*, pp. 107-108, Amsterdam, 1982.

[17] D. E. Knuth, "Literate Programming," Report Number STAN-CS-82-981, Stanford University, Department of Computer Science, 1983.

[18] W. L. Johnson, S. W. Draper, and E. Soloway, "Classifying Bu Is a Tricky Business," *ACM SIGSOFT/SIGPLAN Symposium High-Level Debugging*, Asilomar, CA, March, 1983.

[19] R. W. Root and S. Draper, "Questionnaires as a Softw Evaluation Tool," *Proceedings of the CHI '83 Conference Human Factors in Computing Systems*, pp. 83-87, Boston, 1983.

[20] C. Lewis, "The 'Thinking-Aloud' Method in Interface Evaluation," Tutorial Number 4, presented at the CHI '83 Conference on Human Factors in Computing Systems, Boston, Dec. 1983.

[21] B. Shneiderman, "The Future of Interactive Systems and the Emergence of Direct Manipulation," *Behavior and Information Technology*, 1, pp. 237-256, 1982.

Cognitive Science ONR Technical Report List

8001. Donald R. Gentner, Jonathan Grudin, and Eileen Conway. *Finger Movements in Transcription Typing.* May 1980.

8002. James L. McClelland and David E. Rumelhart. *An Interactive Activation Model of the Effect of Context in Perception: Part I.* May 1980.

8003. David E. Rumelhart and James L. McClelland. *An Interactive Activation Model of the Effect of Context in Perception: Part II.* July 1980.

8004. Donald A. Norman. *Errors in Human Performance.* August 1980.

8005. David E. Rumelhart and Donald A. Norman. *Analogical Processes in Learning.* September 1980.

8006. Donald A. Norman and Tim Shallice. *Attention to Action: Willed and Automatic Control of Behavior.* December 1980.

8101. David E. Rumelhart. *Understanding Understanding.* January 1981.

8102. David E. Rumelhart and Donald A. Norman. *Simulating a Skilled Typist: A Study of Skilled Cognitive-Motor Performance.* May 1981.

8103. Donald R. Gentner. *Skilled Finger Movements in Typing.* July 1981.

8104. Michael I. Jordan. *The Timing of Endpoints in Movement.* November 1981.

8105. Gary Perlman. *Two Papers in Cognitive Engineering: The Design of an Interface to a Programming System and MENUNIX: A Menu-Based Interface to UNIX (User Manual).* November 1981.

8106. Donald A. Norman and Diane Fisher. *Why Alphabetic Keyboards Are Not Easy to Use: Keyboard Layout Doesn't Much Matter.* November 1981.

8107. Donald R. Gentner. *Evidence Against a Central Control Model of Timing in Typing.* December 1981.

8201. Jonathan T. Grudin and Serge Larochelle. *Digraph Frequency Effects in Skilled Typing.* February 1982.

8202. Jonathan T. Grudin. *Central Control of Timing in Skilled Typing.* February 1982.

8203. Amy Geoffroy and Donald A. Norman. *Ease of Tapping the Fingers in a Sequence Depends on the Mental Encoding.* March 1982.

8204. LNR Research Group. *Studies of Typing from the LNR Research Group: The role of context, differences in skill level, errors, hand movements, and a computer simulation.* May 1982.

8205. Donald A. Norman. *Five Papers on Human-Machine Interaction.* May 1982.

8206. Naomi Miyake. *Constructive Interaction.* June 1982.

8207. Donald R. Gentner. *The Development of Typewriting Skill.* September 1982.

8208. Gary Perlman. *Natural Artificial Languages: Low-Level Processes.* December 1982.

8301. Michael C. Mozer. *Letter Migration in Word Perception.* April 1983.

8302. David E. Rumelhart and Donald A. Norman. *Representation in Memory.* June 1983.

8303. The HMI Project at University of California, San Diego. *User Centered System Design: Papers for the CHI '83 Conference on Human Factors in Computer Systems.* November 1983.

8304. Paul Smolensky. *Harmony Theory: A Mathematical Framework for Stochastic Parallel Processing.* December 1983.

8401. Stephen W. Draper and Donald A. Norman. *Software Engineering for User Interfaces.* January 1984.

ICS Technical Report List

8301. David Zipser. *The Representation of Location.* May 1983.

8302. Jeffrey Elman & Jay McClelland. *Speech Perception as a Cognitive Process: The Interactive Activation Model.* April 1983.

8303. Ron Williams. *Unit Activation Rules for Cognitive Networks.* November 1983.

8304. David Zipser. *The Representation of Maps.* November 1983.

8305. The HMI Project. *User Centered System Design: Papers for the CHI '83 Conference on Human Factors in Computer Systems.* November 1983.

8306. Paul Smolensky. *Harmony Theory: A Mathematical Framework for Stochastic Parallel Processing.* December 1983.

8401. Stephen W. Draper and Donald A. Norman. *Software Engineering for User Interfaces.* January 1984.

8402. Steven L. Greenspan. *Reference Comprehension: A Topic-Comment Analysis of Sentence-Picture Verification.* February 1984.

Navy

1 Robert Ahlers
Code N711
Human Factors Laboratory
NAVTRAEQUIPCEN
Orlando, FL 32813

1 Dr. Ed Aiken
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Robert Blanchard
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Nick Bond
Office of Naval Research
Liaison Office, Far East
APO San Francisco, CA 96503

1 Dr. Richard Cantone
Navy Research Laboratory
Code 7510
Washington, DC 20375

1 Dr. Fred Chang
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Stanley Collyer
Office of Naval Technology
800 N. Quincy Street
Arlington, VA 22217

1 CDR Mike Curran
Office of Naval Research
800 N. Quincy St.
Code 270
Arlington, VA 22217

1 Dr. Jude Franklin
Code 7510
Navy Research Laboratory
Washington, DC 20375

1 Dr. Jim Hollan
Code 14
Navy Personnel R & D Center
San Diego, CA 92152

1 Dr. Ed Hutchins
Navy Personnel R&D Center
San Diego, CA 92152

Navy

1 Dr. Norman J. Kerr
Chief of Naval Technical Training
Naval Air Station Memphis (75)
Millington, TN 38054

1 Dr. Peter Kincaid
Training Analysis & Evaluation Group
Dept. of the Navy
Orlando, FL 32813

1 Dr. William L. Maloy (02)
Chief of Naval Education and Training
Naval Air Station
Pensacola, FL 32508

1 Dr. Joe McLachlan
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. James McMichael
Navy PersonnelR&D Center
San Diego, CA 92152

1 Dr William Montague
NPRDC Code 13
San Diego, CA 92152

1 Library, Code P201L
Navy Personnel R&D Center
San Diego, CA 92152

1 Technical Director
Navy Personnel R&D Center
San Diego, CA 92152

6 Commanding Officer
Naval Research Laboratory
Code 2627
Washington, DC 20390

1 Office of Naval Research
Code 433
800 N. Quincy SStreet
Arlington, VA 22217

6 Personnel & Training Research Group
Code 442PT
Office of Naval Research
Arlington, VA 22217

Navy

1 Dr. Gary Poock
Operations Research Department
Code 55PK
Naval Postgraduate School
Monterey, CA 93940

1 Dr. Gil Ricard
Code N711
NTEC
Orlando, FL 32813

1 Dr. Robert G. Smith
Office of Chief of Naval Operations
OP-987H
Washington, DC 20350

1 Dr. Alfred F. Smode, Director
Training Analysis & Evaluation Group
Dept. of the Navy
Orlando, FL 32813

1 Dr. Richard Snow
Liaison Scientist
Office of Naval Research
Branch Office, London
Box 39
FPO New York, NY 09510

1 Dr. Richard Sorensen
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Frederick Steinheiser
CNO - OP115
Navy Annex
Arlington, VA 20370

1 Roger Weissinger-Baylon
Department of Administrative Sciences
Naval Postgraduate School
Monterey, CA 93940

1 Mr John H. Wolfe
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Wallace Wulfeck, III
Navy Personnel R&D Center
San Diego, CA 92152

Marine Corps

1 H. William Greenup
Education Advisor (E031)
Education Center, MCDEC
Quantico, VA 22134

1 Special Assistant for Marine
Corps Matters
Code 100M
Office of Naval Research
800 N. Quincy St.
Arlington, VA 22217

1 DR. A.L. SLAFKOSKY
SCIENTIFIC ADVISOR (CODE RD-1)
HQ, U.S. MARINE CORPS
WASHINGTON, DC 20380

Army

1 Technical Director
U. S. Army Research Institute for the
Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Beatrice J. Farr
U. S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Harold F. O'Neil, Jr.
Director, Training Research Lab
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Commander, U.S. Army Re
for the Behavioral &
ATTN: PERI-SR (Dr. Judi
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Joseph Psotka, Ph.D.
ATTN: PERI-IC
Army Research Institute
5001 Eisenhower Ave.
Alexandria, VA 22333

1 Dr. Robert Sasmor
U. S. Army Research Institute for the
Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

Air Force

1 U.S. Air Force Office of Scientific
Research
Life Sciences Directorate, NL
Bolling Air Force Base
Washington, DC 20332

1 Bryan Dallman
AFHRL/LRT
Lowry AFB, CO 80230

1 Dr. Alfred R. Fregly
AFOSR/NL
Bolling AFB, DC 20332

1 Dr. Genevieve Haddad
Program Manager
Life Sciences Directorate
AFOSR
Bolling AFB, DC 20332

1 Dr. John Tangney
AFOSR/NL
Bolling AFB, DC 20332

1 Dr. Joseph Yasutake
AFHRL/LRT
Lowry AFB, CO 80230

Department of Defense

12 Defense Technical Information Center
Cameron Station, Bldg 5
Alexandria, VA 22314
Attn: TC

1 Military Assistant for Training and
Personnel Technology
Office of the Under Secretary of Defens
for Research & Engineering
Room 3D129, The Pentagon
Washington, DC 20301

1 Major Jack Thorpe
DARPA
1400 Wilson Blvd.
Arlington, VA 22209

1 Dr. Robert A. Wisher
OUSDRE (ELS)
The Pentagon, Room 3D129
Washington, DC 20301

Civilian Agencies

1 Dr. Patricia A. Butler
NIE-BRN Bldg. Stop # 7
1200 19th St., NW
Washington, DC 20208

1 Dr. Susan Chipman
Learning and Development
National Institute of Education
1200 19th Street NW
Washington, DC 20208

1 Dr. Andrew R. Molnar
Office of Scientific and Engineering
Personnel and Education
National Science Foundation
Washington, DC 20550

1 Dr. Everett Palmer
Mail Stop 239-3
NASA-Ames Research Center
Moffett Field, CA 94035

1 Dr. Mary Stoddard
C 10, Mail Stop B296
Los Alamos National Laboratories
Los Alamos, NM 87545

1 Chief, Psychological Research Branch
U. S. Coast Guard (G-P-1/2/TP42)
Washington, DC 20593

1 Dr. Edward C. Weiss
National Science Foundation
1800 G Street, NW
Washington, DC 20550

1 Dr. Frank Withrow
U. S. Office of Education
400 Maryland Ave. SW
Washington, DC 20202

1 Dr. Joseph L. Young, Director
Memory & Cognitive Processes
National Science Foundation
Washington, DC 20550

Private Sector

1 Dr. John R. Anderson
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

1 Dr. Alan Baddeley
Medical Research Council
Applied Psychology Unit
15 Chaucer Road
Cambridge CB2 2EF
ENGLAND

1 Eva L. Baker
Director
UCLA Center for the Study of Evaluation
145 Moore Hall
University of California, Los Angeles
Los Angeles, CA 90024

1 Mr. Avron Barr
Department of Computer Science
Stanford University
Stanford, CA 94305

1 Dr. Menucha Birenbaum
School of Education
Tel Aviv University
Tel Aviv, Ramat Aviv 69978
Israel

1 Dr. John Black
Yale University
Box 11A, Yale Station
New Haven, CT 06520

1 Dr. John S. Brown
XEROX Palo Alto Research Center
3333 Coyote Road
Palo Alto, CA 94304

1 Dr. Glenn Bryan
6208 Poe Road
Bethesda, MD 20817

1 Dr. Bruce Buchanan
Department of Computer Science
Stanford University
Stanford, CA 94305

1 Dr. Jaime Carbonell
Carnegie-Mellon University
Department of Psychology
Pittsburgh, PA 15213

Private Sector

1 Dr. Pat Carpenter
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

1 Dr. Micheline Chi
Learning R & D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

1 Dr. William Clancey
Department of Computer Science
Stanford University
Stanford, CA 94306

1 Dr. Michael Cole
University of California
at San Diego
Laboratory of Comparative
Human Cognition - D003A
La Jolla, CA 92093

1 Dr. Allan M. Collins
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

1 Dr. Emmanuel Donchin
Department of Psychology
University of Illinois
Champaign, IL 61820

1 ERIC Facility-Acquisitions
4833 Rugby Avenue
Bethesda, MD 20014

1 Dr. Anders Ericsson
Department of Psychology
University of Colorado
Boulder, CO 80309

1 Mr. Wallace Feurzeig
Department of Educational Technology
Bolt Beranek & Newman
10 Moulton St.
Cambridge, MA 02238

1 Dr. Dexter Fletcher
University of Oregon
Department of Computer Science
Eugene, OR 97403

Private Sector

1 Dr. John R. Frederiksen
Bolt Beranek & Newman
50 Moulton Street
Cambridge, MA 02138

1 Dr. Michael Genesereth
Department of Computer Science
Stanford University
Stanford, CA 94305

1 Dr. Don Gentner
Center for Human Information Processing
University of California, San Diego
La Jolla, CA 92093

1 Dr. Dedre Gentner
Bolt Beranek & Newman
10 Moulton St.
Cambridge, MA 02138

1 Dr. Robert Glaser
Learning Research & Development Center
University of Pittsburgh
3939 O'Hara Street
PITTSBURGH, PA 15260

1 Dr. Joseph Goguen
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

1 Dr. Daniel Gopher
Faculty of Industrial Engineering
& Management
TECHNION
Haifa 32000
ISRAEL

1 DR. JAMES G. GREENO
LRDC
UNIVERSITY OF PITTSBURGH
3939 O'HARA STREET
PITTSBURGH, PA 15213

1 Dr. Barbara Hayes-Roth
Department of Computer Science
Stanford University
Stanford, CA 95305

1 Dr. Frederick Hayes-Roth
Teknowledge
525 University Ave.
Palo Alto, CA 94301

Private Sector

1 Dr. Joan I. Heller
Graduate Group in Science and
Mathematics Education
c/o School of Education
University of California
Berkeley, CA 94720

1 Dr. James R. Hoffman
Department of Psychology
University of Delaware
Newark, DE 19711

1 American Institutes for Research
1055 Thomas Jefferson St., N.W.
Washington, DC 20007

1 Dr. Kristina Hooper
Corporate Research, ATARI
1196 Borregas
Sunnyvale, CA 94086

1 Dr. Earl Hunt
Dept. of Psychology
University of Washington
Seattle, WA 98105

1 Robin Jeffries
Computer Research Center
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304

1 Dr. Marcel Just
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

1 Dr. David Kieras
Department of Psychology
University of Arizona
Tucson, AZ 85721

1 Dr. Walter Kintsch
Department of Psychology
University of Colorado
Boulder, CO 80302

1 Dr. Stephen Kosslyn
1236 William James Hall
33 Kirkland St.
Cambridge, MA 02138

Private Sector

1 Dr. Pat Langley
The Robotics Institute
Carnegie-Mellon University
Pittsburgh, PA 15213

1 Dr. Jill Larkin
Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213

1 Dr. Alan Lesgold
Learning R&D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15260

1 Dr. Jim Levin
University of California
at San Diego
Laboratory for Comparative
Human Cognition - D003A
La Jolla, CA 92093

1 Dr. Michael Levine
Department of Educational Psychology
210 Education Bldg.
University of Illinois
Champaign, IL 61801

1 Dr. Marcia C. Linn
Lawrence Hall of Science
University of California
Berkeley, CA 94720

1 Dr. Don Lyon
AFHRL/OT (UDRI)
Williams AFB, AZ 85225

1 Dr. Jay McClelland
Department of Psychology
MIT
Cambridge, MA 02139

1 Dr. James R. Miller
Computer*Thought Corporation
1721 West Plano Highway
Plano, TX 75075

1 Dr. Mark Miller
Computer*Thought Corporation
1721 West Plano Parkway
Plano, TX 75075

Private Sector

1 Dr. Tom Moran
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

1 Dr. Allen Munro
Behavioral Technology Laboratories
1845 Elena Ave., Fourth Floor
Redondo Beach, CA 90277

1 Dr. Donald A Norman
Cognitive Science, C-015
Univ. of California, San Diego
La Jolla, CA 92093

1 Dr. Jesse Orlansky
Institute for Defense Analyses
1801 N. Beauregard St.
Alexandria, VA 22311

1 Dr. Nancy Pennington
University of Chicago
Graduate School of Business
1101 E. 58th St.
Chicago, IL 60637

1 DR. PETER POLSON
DEPT. OF PSYCHOLOGY
UNIVERSITY OF COLORADO
BOULDER, CO 80309

1 Dr. Lynn Reder
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

1 Dr. Fred Reif
Physics Department
University of California
Berkeley, CA 94720

1 Dr. Lauren Resnick
LRDC
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 1521

1 Dr. Jeff Richardson
Denver Research Institute
University of Denver
Denver, CO 80208

Private Sector

1 Mary S. Riley
Program in Cognitive Science
Center for Human Information Processing
University of California, San Diego
La Jolla, CA 92093

1 Dr. Andrew M. Rose
American Institutes for Research
1055 Thomas Jefferson St. NW
Washington, DC 20007

1 Dr. Ernst Z. Rothkopf
Bell Laboratories
Murray Hill, NJ 07974

1 Dr. William B. Rouse
Georgia Institute of Technology
School of Industrial & Systems
Engineering
Atlanta, GA 30332

1 Dr. David Rumelhart
Center for Human Information Processing
Univ. of California, San Diego
La Jolla, CA 92093

1 Dr. Michael J. Samet
Perceptronics, Inc
6271 Variel Avenue
Woodland Hills, CA 91364

1 Dr. Roger Schank
Yale University
Department of Computer Science
P.O. Box 2158
New Haven, CT 06520

1 Dr. Walter Schneider
Psychology Department
603 E. Daniel
Champaign, IL 61820

1 Dr. Alan Schoenfeld
Mathematics and Education
The University of Rochester
Rochester, NY 14627

1 Mr. Colin Sheppard
Applied Psychology Unit
Admiralty Marine Technology Est.
Teddington, Middlesex
United Kingdom

Private Sector

1 Dr. M. Wallace Sinaiko
Program Director
Manpower Research and Advisory Services
Smithsonian Institution
801 North Pitt Street
Alexandria, VA 22314

1 Dr. Edward E. Smith
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

1 Dr. Elliott Soloway
Yale University
Department of Computer Science
P.O. Box 2158
New Haven, CT 06520

1 Dr. Kathryn T. Spoehr
Psychology Department
Brown University
Providence, RI 02912

1 Dr. Robert Sternberg
Dept. of Psychology
Yale University
Box 11A, Yale Station
New Haven, CT 06520

1 Dr. Albert Stevens
Bolt Beranek & Newman, Inc.
10 Moulton St.
Cambridge, MA 02238

1 David E. Stone, Ph.D.
Hazeltime Corporation
7680 Old Springhouse Road
McLean, VA 22102

1 Dr. Perry W. Thorndyke
Perceptronics, Inc.
545 Middlefield Road, Suite 140
Menlo Park, CA 94025

1 Dr. Douglas Towne
Univ. of So. California
Behavioral Technology Labs
1845 S. Elena Ave.
Redondo Beach, CA 90277

1 Dr. Kurt Van Lehn
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

Private Sector

1 Dr. Keith T. Wescourt
Perceptronics, Inc.
545 Middlefield Road, Suite 140
Menlo Park, CA 94025

1 William B. Whitten
Bell Laboratories
2D-610
Holmdel, NJ 07733

1 Dr. Thomas Wickens
Department of Psychology
Franz Hall
University of California
405 Hilgarde Avenue
Los Angeles, CA 90024

1 Dr. Mike Williams
IntelliGenetics
124 University Avenue
Palo Alto, CA 94301

1 Dr. Joseph Wohl
Alphatech, Inc.
2 Burlington Executive Center
111 Middlesex Turnpike
Burlington, MA 01803

END

DATE
FILMED

4 -84

DTIC